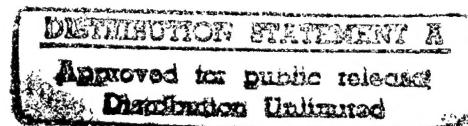


DARPA Year 2000 Plan of Action (DRAFT of 11/22/96)

1997

Contents

- Executive Summary
- Tools for Static Analysis and Renovation
- Tools for Dynamic Analysis
- Visualization
- Contingency Fund



Executive Summary

The Year 2000 Problem (Y2K) is so severe because it is both pervasive and the causes are idiosyncratic.

- Computer systems rely on operating systems and a variety of support packages, and they may interact with numerous other computer systems. Because a system is only as strong as its weakest link, a Y2K problem in any of the auxiliary systems could cause as much trouble as a direct Y2K problem in the primary system.
- Dates are stored and calculated in many different formats. (Over 120 date standards exist, and countless ad hoc ones have been devised according to the whims of generations of mostly undisciplined programmers.)

Because of the latter issue, the Y2K problem cannot be tackled completely through automated means; however, because of the pervasiveness of the problem, the cost will be out of hand unless technology is brought to bear on the problem. Technology can reduce the cost of addressing the Y2K problem by automating tasks related to:

- Identifying code that will work incorrectly post-year-2000.
- "Date prospecting": Tracing the processing of date-related information to determine if problems exist and to identify code or data that needs to be changed.
- Transforming the necessary code and data.
- Testing the changes -- both in individual programs and across groups of programs (e.g., all programs that access a data file in which dates have been converted from 2 digits to 4 digits).
- Managing the process -- especially how to synchronize the changes in separate programs/systems.

In keeping with its role as the principal Agency within the DoD for innovative, high-payoff research and development projects, DARPA will direct its efforts on ways to bring technology to bear on the DoD's Y2K problem. The project has three major focus areas:

Tools for static analysis and renovation:

19971204 156

DTIC QUALITY INSPECTED 4

The retargeting of the kinds of tools being used to address the Y2K problem in Cobol programs to the languages used in DoD command-and-control and weapon systems, namely, Ada, C, Fortran, Jovial, and CMS-2.

Tools for dynamic analysis:

The creation of tools that apply information obtained from test runs of programs to detect some of the sites of problematic date-manipulation code. In addition to the application to "date prospecting", this approach also has the potential to help out with two other important Y2K problems: (i) determining whether COTS components have date problems, and (ii) testing renovated code. This approach also represents one of the few methods that is able to identify date-manipulation problems in programs for which we do not have source code.

Visualization:

The application of "program-visualization techniques" -- the use of interactive color graphics to provide multiple, linked views of program properties and aggregate statistics about programs -- to provide improved user-interfaces to Y2K tools.

The latter two areas involve the transition to industry of certain technologies from the research community that go beyond present commercial Y2K products and services. They represent the application of technology not normally associated with identifying possible Y2K problems, but have the potential to have immediate impact on the Y2K problem in DoD systems.

To reduce risk, multiple teams will be funded (e.g., to retarget their Cobol Y2K technology to Ada, C, Fortran, Jovial, and CMS-2), and multiple technology approaches will be followed (e.g., approaches for gathering test information by "instrumenting" object code will be funded along with approaches for instrumenting source code.)

An important aspect of each of the funded activities will be the early application of the tools in appropriate pilot projects in the DoD so that an assessment can be made of how well they work.

DARPA's project will be complementary to (and more pro-active than) the more management-oriented efforts being carried out within the DoD (for example, inventorying of commercially available technology and COTS compliance -- e.g., DISA -- and certification of tools and renovation techniques -- e.g., Hill AFB, STSC Reengineering Group .

Time is of the essence. Given the stated (albeit unrealistic) goal of the government that all systems are to be renovated by Jan. 1, 1999 (so as to provide a full-year cushion for testing of renovated systems), for the project to have any impact, its artifacts must be in the marketplace by the end of 1997. However, DARPA's role will also continue past the end of 1997: To maximize the impact of the project's products on DoD projects, it will be necessary to ensure that an adequate level of technical expertise (e.g., in the way of training and technical support) is available post-1997.

Finally, because this is a crash project with a short deadline, the program manager will be supplied with a contingency fund to give him the opportunity to pursue ideas on short notice that may be generated during the course of the project.

Cost Breakdown

Tools for static analysis and renovation:

104 staff-months/team x 3 teams = 312 staff-months

Tools for dynamic analysis:

Source-code-instrumentation approach: 34 staff-months + 54 staff-months = 88 staff-months

Object-code-instrumentation approach: 34 staff-months + 136 staff-months = 170 staff-months

Visualization:

30 staff-months

Post-1997 support:

72 staff-months

Contingency fund:

30 staff-months

Total Estimated Cost

702 staff-months x \$20,000/staff-month = \$14,040,000

Caveats

- The figure of \$20,000/staff-month includes fringe benefits, office space, equipment, and travel. This seems not unreasonable, given the skill level of personnel we need to attract and the need to equip them with high-powered workstations equipped with large amounts of memory.
- The overall figure of \$14M represents a 12-month crash effort (involving 58.5 staff-years, 10% of which is post-1997 follow-on technical support). If this sounds high, bear in mind that what is detailed below represents a plan for a comprehensive technical effort by DARPA to bring appropriate technology to bear on the DoD Y2K problem. It would be possible to scale back in a number of ways if it were deemed that DARPA's efforts should just lay the groundwork (i.e., to show that these approaches work), letting the services then take over. If this turns out to be the case, then numbers like the number of language dialects to be supported, the number of hardware/OS platforms to be supported, etc. should be rethought.
- Not all parts of the plan represent the same level of risk (in terms of expected success/failure):
 - The material proposed in the Visualization category is definitely doable.
 - The material in the Tools for Static Analysis and Renovation category is somewhat risky (because of the ramping-up and time-frame issues), but doable.
 - The material in the Tools for Dynamic Analysis category is quite risky, but possibly very high payoff.

Overall, the need to find 53+ highly qualified people to work on on this in 1997 represents a substantial risk, in and of itself.

Tools for Static Analysis and Renovation

The Opportunity

An important goal of the project is to ensure that the kind of tools being used to address the Y2K problem in Cobol programs are retargeted to the languages used in DoD command-and-control and weapons systems, namely, Ada, C, Fortran, Jovial, CMS-2. The capabilities of commercially available Cobol Y2K tools include:

- Detection of problematic date fields, date literals, and date-manipulation code. This may involve data items holding past dates, future dates, or date intervals.
- Impact analysis (a.k.a. program slicing): Display of potential "ripple-effects" -- through source code, JCL, and databases or files -- that would occur if elements of the program were changed.
- Transformation of problematic date fields, date literals, and date-manipulation code (either by insertion of compensating operations in date-related comparisons and calculations, or by replacement of date-manipulation code with calls to libraries of standard date operations). Support for "expansion" (i.e., transformation of 2-digit year fields to 4-digit fields), "windowing", and "encoding" (e.g., YYMMDD -> CYYDDD) renovation strategies.
- Conversion of date fields in dialog screens and reports.
- Conversion of various types of databases and file formats.
- Change synchronization: history management, configuration management, and management of hybrid systems (i.e., coexistence of converted and unconverted subsystems).
- Conversion of JCL to support conversion results and to support hybrid coexistence.
- Maintenance of converted and unconverted values in mirrored databases and data files during hybrid coexistence.

Fortunately, many of these tasks can be implemented in a language-independent manner. That is, tools can be structured so that renovation tasks are implemented in terms of a common base of data structures that can be used to represent programs written in different programming languages.

Work to be Carried Out

(Note: The cost estimates for the following projects are those for a single team.)

- Development of front ends for parsing Ada, C, and (multiple dialects of) Fortran, Jovial, and CMS-2, as well as the development of "pretty-printers" for each language. (For purposes of this discussion, "parsing" includes the task of building abstract-syntax trees (ASTs) and symbol tables (STs). Pretty-printers handle the redisplay of ASTs as text; they are necessary to convert renovated ASTs -- an internal data structure -- into renovated source text.) We can probably assume that each contractor already has front ends for at least two of the languages. [*Estimated cost*: 8 staff-months/language x 3 languages + 1.5 staff-months/dialect x 8 dialects = 36 staff-months.]
- Implementation of generalized versions of existing internal structures of Cobol Y2K tool (i.e., symbol tables, control-flow graphs, call graphs, and dependence graphs) to create a common base of language-independent internal structures. Generalization of renovation operations to work on the common base of language-independent internal structures. [*Estimated cost*: 12 staff-months]
- Implementation of translation procedures from Ada, C, Fortran, Jovial, and CMS-2 AST+ST structures to the common base of language-independent internal structures (i.e., control-flow graphs, call graphs, and dependence graphs). [*Estimated cost*: 6 staff-months/language x 5 languages + 1 staff-month/dialect x 8 dialects = 38 staff-months.]
- Pilot projects. [*Estimated cost*: 4 staff-months/project x 4 pilot projects = 16 staff-months.]

Possible Participants/Contractors

- Reasoning Systems
- Xinotech Research
- The Software Revolution, Inc.
- Robert Filman (Lockheed R&D Division, Palo Alto, CA)

- Susan Roberts and Howard Reubenstein (MITRE)
 - David S. Warren (State University of New York at Stony Brook)
-

Tools for Dynamic Analysis

The Opportunity

Information obtained via test runs of appropriately instrumented programs can be used to detect some of the sites of problematic date-manipulation code. In addition to the application to "date prospecting", they also have the potential to help out with two other important Y2K problems: (i) determining whether COTS components have date problems, and (ii) testing renovated code.

These techniques represent the application of technology not normally associated with identifying possible Y2K problems, and are one of the few methods we are aware of for identifying date-manipulation problems in programs for which we do not have source code.

There are two techniques that can be used to identify candidate sites of date-manipulation problems:

Detection of pointer- and array-access errors:

By detecting "index-out-of-bounds" (and/or "pointer-out-of-bounds") errors that occur during an execution run of a program on post-year-2000 data, some kinds of erroneous date manipulations can be detected.

Use of path profiles to detect date-dependent branches:

In path profiling, the number of times each different (loop-free) path executes is accumulated during an execution run. Thus, with such an instrumented program, each run (or set of runs) of the program generates a "path spectrum" for the execution -- a distribution of the paths that were executed. Path profiles can be used to identify sites in a program that are good candidates for being date-dependent branches. To do this, one would obtain path spectra from execution runs on pre-year-2000 data and post-year-2000 data (or whatever "date vulnerability" we are trying to test). By comparing the two path spectra, paths that indicate where the program veered into "uncharted territory" on the post-year-2000 runs (as well as paths that were no longer executed on the post-year-2000 runs) can be identified. (With some further analysis of such paths, this information can also be used to identify specific locations where date-dependent branches might have occurred.)

The techniques also help with the problems of testing COTS components and testing renovated code. For example, a correctly renovated system should have similar path spectra from execution runs on pre-year-2000 data and post-year-2000 data; path-spectrum differences could indicate remaining problems.

After such techniques have been used to identify candidate sites of problems, this information can be "amplified" --- via searching and slicing operations --- to find other potential locations of problems. In addition to the *static* impact-analysis (slicing) techniques that are supported in some Cobol Y2K tools, instrumentation techniques (of a third kind, different from the two described above) can also be used to support a different kind of slicing operation: *dynamic slicing*. Dynamic slicing uses data about the

dynamic dependences that are actually exhibited during sample runs of the program:

Dynamic slicing:

The program is instrumented to keep track, for each memory location, of what site in the program most recently wrote to it. A read performed at site t of a memory location last written by site s reveals a dynamic dependence of t on s . When the program is executed, the dynamic dependences are recorded.

This information is useful for purposes of Y2K renovation because the record of dynamic dependences from a run of the program can be used to recover the way date information actually "flows" from point to point during that run. After date-problem discovery techniques (e.g., pattern matching, or the techniques described above based on catching access errors and path profiling) have been applied to identify candidate sites of problems, dynamic slicing can be used to identify other sites at which related date manipulations were carried out.

Static slicing and dynamic slicing are complementary operations: Static slicing is based on a conservative dependence analysis of the source code; however, in programs in which there is heavy usage of pointers and aliasing, it is sometimes grossly inaccurate. Because dynamic slicing reports "precise" information about actual dependences, it is less likely to lead to information overload. (Note, however, that dynamic slicing is "precise" only for the dependences actually exhibited by a run of the program. It will fail to report dependences that might occur when the program is run with different input data.)

With all three of the techniques described above, an algorithm is applied to instrument the program in an appropriate way. Instrumentation can be carried out at a number of different levels:

- on the source code;
- on "intermediate representations" of the kind used in compilers (such as a program's control-flow graph);
- on object-code files (such as UNIX ".o" files);
- on executable files (such as UNIX "a.out" files).

Work to be Carried Out

A two-pronged approach will be carried out: one based on source-code instrumentation, and one based on object-code/machine-code instrumentation.

- In source-code instrumentation, a source-to-source transformation is performed to insert the instrumentation instructions into the source program, which is then recompiled and run.
- With object-code or machine-code instrumentation, the instrumentation instructions are inserted into the low-level code. This necessitates either sliding code to permit instrumentation instructions to be inserted in place, or inserting jumps to "patch space", where instrumentation actions are actually performed.

Instrumentation of object-code or machine-code has the advantage that a single instrumentation system can apply to systems written in a mixture of different source languages. The disadvantage is that DoD systems run on every conceivable kind of hardware/OS platform, and a low-level interface involving instruction-set formats and/or object-code formats must be written for each hardware/OS platform to be

supported.

Some work items are common to both approaches (although their ultimate realization in the two approaches may be different):

- Implement a "path-spectrum" comparator that identifies locations of possible date-dependent branches by comparing two or more path spectra. [*Estimated cost*: 6 staff-months]
- Presenting all path-spectrum differences to users is likely to overwhelm them with far too much information. The information should be filtered or summarized in some fashion. The challenge here is to devise ways of weighting the path-spectrum differences so that higher-weight items are more likely to correspond to the most important sites of problematic date manipulations. For example, it may be that path-spectrum differences that involve paths generated early on in one of the runs are more likely to be of interest to the user. These would identify the initial occurrences of date-dependent branching, and the user could then use slicing operations to find other related date manipulations. (This may require extensions to the basic instrumentation technique in order to record information about the order in which paths are generated.) [*Estimated cost*: 12 staff-months]
- Pilot projects. [*Estimated cost*: 4 staff-months/project x 4 pilot projects = 16 staff-months.]

Work Items: Source-Code Instrumentation

The approach based on instrumenting source code is related to the effort on tools for static analysis and renovation. The parsers, abstract-syntax trees, symbol tables, control-flow graphs, and unparsers developed in that part of the project provide the great majority of the infrastructure needed for source-code instrumentation.

- Implement the appropriate source-level instrumentation algorithms. [*Estimated cost*: 18 staff-months/instrumentation technique x 3 techniques = 54 staff-months.]

Although the contractor who performs the source-code-instrumentation work need not be the same as the one building tools for the "Static Analysis and Renovation" effort, they will have to work in close concert.

Possible Participants/Contractors

- Reasoning Systems
- Xinotech Research
- The Software Revolution, Inc.
- Thomas Reys (University of Wisconsin)
- Tim Teitelbaum (Cornell University and GrammaTech, Inc.)
- Thomas Ball (Lucent Technologies, Naperville, IL)
- Reliable Software Technologies
- Other testing companies

Work Items: Object-Code Instrumentation

- For each instrumentation technique, there are two possibilities:
 1. Implement a system to instrument object-code that is designed to make it possible to be

ported easily to different platforms, and carry out ports to the appropriate platforms. [Estimated cost: 24 staff-months + 4 staff-months/platform x 12 platforms = 72 staff-months.]

2. Implement a processor that generates object-code instrumentation systems for different machines from descriptions of hardware instructions sets and object-code-file (or machine-code-file) formats, and develop instruction-set descriptions and object-code descriptions for appropriate platforms. [Estimated cost: 36 staff-months + 2 staff-months/platform x 12 platforms = 60 staff-months.]
- Preliminary pilot project: Test out the idea of using path-profiling to find date-dependent branches using the existing Wisconsin path-profiling system (which runs on the Sun/Solaris platform). Alternatively, port the Wisconsin system to the RS-6000/AIX platform so as to be able to test out the idea on a system such as the one at the Naval Space Command. [Estimated cost: 4 staff-months]

(Cavaet: The commercialization path for the approach based on object-code instrumentation remains to be worked out.)

Possible Participants/Contractors

- James Larus (University of Wisconsin)
- Thomas Ball (Lucent Technologies, Naperville, IL)
- Mary Fernandez (AT&T Laboratories)
- Norman Ramsey (University of Virginia)
- Thomas Reps (University of Wisconsin)
- David Wall (Silicon Graphics)
- Jay Lapreau (University of Utah)
- Reinhard Wilhelm (Universitaet des Saarlandes, Saarbruecken, Germany)
- BellCore dynamic-slicing people
- Reliable Software Technologies
- Other testing companies

Visualization

The Opportunity

A segment of the research community has been investigating "program-visualization techniques" -- the use of interactive color graphics to provide multiple, linked views of program properties and aggregate statistics about programs. Typically, these visualization systems use color, spatial layout, and "brushing" (i.e., highlighting relevant portions or projections of the information as the user moves the mouse through the display) to provide ways of exploring program properties along multiple dimensions. The views of the program can also be used for navigation: Selection operations and zooming operations on the view provide access mechanisms to the code (allowing the user to "drill in" to the code from the high-level view).

These visualization techniques are not used in present Y2K tools, but could be used to display information about possible date-manipulation locations obtained via

- searching
- testing instrumented code
- slicing (either static and dynamic)
- accessing the program's version history (in order to identify other parts of the "same feature" as date-manipulation code)

A second opportunity for exploiting visualization techniques in Y2K tools is to use them for displaying test results in such distributed systems, for example, for displaying aggregate statistics on communication events. This is relevant to the DoD's Y2K problem because most of DoD's command and control systems are structured as distributed message-passing systems.

Work to be Carried Out

- Some program-visualization technology is just becoming commercially available in the form of packages that map certain file formats to certain kinds of screen displays. A cost-effective way to bring program visualization to bear on the DoD's Y2K problem would be to license the technology from such a vendor (e.g., Lucent Technologies) and perform the necessary integration work with the tools for static analysis and renovation that are to be developed. [*Estimated cost*: 6 staff-months]
- Develop ways of displaying aggregate statistics about communication events gathered during testing of distributed systems. [*Estimated cost*: 24 staff-months]

Possible Participants/Contractors

- Thomas Ball (Lucent Technologies, Naperville, IL)
- Steve Eick (Lucent Technologies, Naperville, IL)
- Dan Fyoch (Lucent Technologies, Naperville, IL)
- Jim Weichel (Lucent Technologies, Naperville, IL)

Contingency Fund

The Opportunity

Because this is a crash project with a short deadline, the program manager will be supplied with a contingency fund to give him the flexibility to be opportunistic -- to pursue ideas on short notice that may be generated during the course of the project, either by contractors being supported by the project or by other members of the DARPA community. Some of the areas in which contingency funds might be applied are:

- Additional methods for testing renovated code
- Other methods for testing of COTS components
- Improved static-slicing methods (e.g., in the presence of aliasing)
- Wrapper technology
- Roll-forward/roll-back methods

- Disassembly

(Some opportunities to address new problems may come to light as the projects described in previous sections take form. For example, there may be opportunities to apply data structures, components, or tools developed for the activities described above in new ways.)

[*Estimated cost*: 30 staff-months]
